Pixel: Heap Buffer Overflow

Sagi Kedmi

Februrary 4, 2016

Contents

1 Synopsis

`Pixel`'s kernel (`msm` kernel tree; `android-msm-marlin-3.18-nougat-mr1` branch) exposes a character device (`/dev/touch_fwu`, handles the touchscreen controller firmware update) that allows a privileged attacker to trigger a heap buffer overflow.

The vulnerability was verified on:

```
sailfish:/ $ getprop ro.build.fingerprint
google/sailfish/sailfish:7.1.1/NMF26U/3562008:user/release-keys
```

2 Heap Buffer Overflow

2.1 Vulnerable Code

The following `file_operations` are registered [1]:

```
static const struct file_operations touch_fwu_fops = {
    [...]
    .write          = touch_fwu_write,
    .unlocked_ioctl = touch_fwu_ioctl,
    .open           = touch_fwu_open,
    [...]
};
```

On `ioctl` syscall, `touch_fwu_ioctl` is called [2]:

```c
static long touch_fwu_ioctl(struct file *filp, unsigned int cmd, unsigned long args)
{
    [...]
    u32 data = 0;
    struct cdev_data *fw_cdev = filp->private_data;
    unsigned char *buf;
    struct firmware *fw;
    [...]
    switch (cmd) {
        [...]
        case FW_FILE_SIZE:
            data = args;
            fwu_data->fw_size = data;
            fw_cdev->fw_size = data;
            pr_info("%s: FW_FILE_SIZE:%d", __func__, data);
            break;
        case FW_FILE_REQUEST:
            if (fw_cdev->fw_size) {
                if (fw_cdev->buf == NULL) {
                    pr_info("%s: allocate buf", __func__);
                    buf = kzalloc(
                        fw_cdev->fw_size*sizeof(unsigned char),
                        GFP_KERNEL);
                    if (!buf) {
                        pr_err("%s, allocate failed", __func__);
                        return -1;
                    }
                    fw_cdev->buf = buf;
                }
            }
            pr_info("%s: FW_FILE_REQUEST", __func__);
            break;
        [...]
}
```

The `FW_FILE_SIZE` `ioctl` command allows an arbitrary size to be specified from user mode and be saved to `fw_cdev->fw_size`.

The `FW_FILE_REQUEST` `ioctl` command simply allocates a kernel heap buffer `fw_cdev->buf` of size `fw_cdev->fw_size`.

On `write` syscall, `touch_fwu_write` is called [3]:

```c
static ssize_t touch_fwu_write(struct file *file, const char __user *buf, size_t count, loff_t *offs
{
    struct cdev_data *fw_cdev = file->private_data;
    u16 *tmp;
    [...]
    tmp = kzalloc(count, GFP_KERNEL);
    [...]
    if(copy_from_user(tmp, buf, count)) {
        [...]
    }
    memcpy(fw_cdev->buf+fw_cdev->size_count, tmp, count);
    [...]
}
```

Since `count` is controllable from user mode, on `write` syscall, an arbitrarly-sized buffer, `tmp`, is securely copied from userspace. Then, `memcpy` is used to copy `count` bytes from `tmp` to `fw_cdev->buf` (`size_count` is zero).

An attacker can simply use the `ioctl` commands to allocate a small heap buffer to `fw_cdev->buf` and use `write` to overrun it with a larger buffer.

2.2 Proof of Concept

In the attached `zip` archive there are both the source `poc.c` and the `aarch64 ELF` binary `poc`.

The source file was compiled with:

```
$ aarch64-linux-gnu-gcc -static poc.c -o poc
```

Try the crasher on a device (you can impersonate the correct `SELinux` context and execute using it, we decided to do it with `su`):

```
$ adb push poc /data/local/tmp
$ adb shell
sailfish:/ $ su
sailfish:/ # cd /data/local/tmp
sailfish:/data/local/tmp # ./poc
```

2.3 Crash Dump

After the device crashes, `/sys/fs/pstore/console-ramoops` has the crash-dump:

```
[43964.066579] c0   6919 Unable to handle kernel paging request at virtual address 6161616161616161
[43964.066652] c0   6919 pgd = ffffffc031d7f000
[43964.066686] [6161616161616161] *pgd=0000000000000000, *pud=0000000000000000
[43964.066761] c0   6919 Internal error: Oops: 96000004 [#1] PREEMPT SMP
[43964.066819] c0   6919 CPU: 0 PID: 6919 Comm: <-transport Tainted: G        W      3.18.31-g226daf
[43964.066853] c0   6919 Hardware name: HTC Corporation. MSM8996pro v1.1 + PMI8996 Sailfish A (DT)
[43964.066889] c0   6919 task: ffffffc057554800 ti: ffffffc0f3590000 task.ti: ffffffc0f3590000
[43964.066955] c0   6919 PC is at __kmalloc+0xc8/0x224
[43964.066990] c0   6919 LR is at __kmalloc+0x94/0x224
[43964.067020] c0   6919 pc : [<ffffffc00019baa4>] lr : [<ffffffc00019ba70>] pstate: 80000145
[43964.067047] c0   6919 sp : ffffffc0f3593c90
[43964.067074] x29: ffffffc0f3593c90 x28: ffffffc001751000
[43964.067126] x27: 00000000fcd8c000 x26: 6161616161616161
[43964.067174] x25: ffffffc0f3593c90 x24: 00000000000000d0
[43964.067223] x23: 000000000000c801c x22: ffffffc0f3590000
[43964.067270] x21: ffffffc00153d5a0 x20: ffffffc0f9801b00
[43964.067318] x19: 0000000000000400 x18: 000000000000002c
[43964.067364] x17: 0000000000000000 x16: ffffffc0001a411c
[43964.067410] x15: 00000000004e4eec x14: 0000000000000dd5
[43964.067457] x13: 2e8ba2e8ba2e8ba3 x12: 0000000000000031
[43964.067503] x11: 0000000000000000 x10: 0000000000000001
[43964.067549] x9 : 0000000000040000 x8 : 000000000000003f
[43964.067596] x7 : 0000000000000000 x6 : ffffffc0f6e41630
[43964.067641] x5 : 0000000000000000 x4 : ffffffc0f3593c80
[43964.067686] x3 : 0000000000000000 x2 : 0000000000000001
[43964.067731] x1 : ffffffc0f3590000 x0 : 0000000000000000
[...]
```

File `1.crash` contains the entire crash-dump.

3 Attack Surface

3.1 DAC

DAC-wise, who can `ioctl`, `write` and `open` the `/dev/touch_fwu` character device?

The attacker has to execute code `UID root` within `device SELinux` context.

```
1|sailfish:/dev # ls -lZ touch_fwu
crw------- 1 root root u:object_r:device:s0 10,  96 1970-01-31 10:14 touch_fwu
```

3.2 SELinux

SELinux-wise, what contexts can `ioctl` and `write` to a `device`? noindent
Looking at the aforementioned DAC, we need to find `SELinux` domains with `allow` rules that have target type `device` with the `ioctl` and `write` permissions on `chr_file` class.

Analysing `Nexus 5x`'s `sepolicy` (NRD90W) yields:

```
allow init device:chr_file { setattr read lock getattr write ioctl open append };
allow ueventd device:chr_file { read lock getattr write ioctl open append };
```

3.3 Processes

What active processes can trigger the vulnerability?

We simply need to find which processes execute with `UID root` within the aforementioned `SELinux` contexts.

Analysing active processes using `ps -Z` yields:

```
u:r:init:s0     root      1     0     10632  1672  SyS_epoll_ 00004ca7d0 S /init
u:r:ueventd:s0  root      395   1     6008   1368  poll_sched 00004ca800 S /sbin/ueventd
```

Code execution within any of the processes above can exploit the vulnerability.

References

[1] `android-msm-marlin-3.18-nougat-mr1` branch. texttt/dev/touch_fwu file operations. https://android.googlesource.com/kernel/msm/+/android-msm-marlin-3.18-nougat-mr1/drivers/input/touchscreen/touch_fw_update.c#460. [Online; accessed 04-February-2016].

[2] `android-msm-marlin-3.18-nougat-mr1` branch. `ioctl` syscall handling. https://android.googlesource.com/kernel/msm/+/android-msm-marlin-3.18-nougat-mr1/drivers/input/touchscreen/touch_fw_update.c#228. [Online; accessed 04-February-2016].

[3] `android-msm-marlin-3.18-nougat-mr1` branch. `write` syscall handling. https://android.googlesource.com/kernel/msm/+/android-msm-marlin-3.18-nougat-mr1/drivers/input/touchscreen/touch_fw_update.c#199. [Online; accessed 04-February-2016].